



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/024,961	12/19/2001	David Pociu	13729-003001	3454
26161	7590	11/14/2005	EXAMINER	
FISH & RICHARDSON PC P.O. BOX 1022 MINNEAPOLIS, MN 55440-1022			MOFIZ, APU M	
			ART UNIT	PAPER NUMBER
			2165	
DATE MAILED: 11/14/2005				

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No.

10/024,961

Applicant(s)

POCIU, DAVID

Examiner

Apu M. Mofiz

Art Unit

2165

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 12 September 2005.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1,4,6-11,13-17,20,21,23,24,26 and 27 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1,4,6-11,13-17,20,21,23,24,26 and 27 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 19 December 2001 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
 2. ☐ Certified copies of the priority documents have been received in Application No. _____.
 3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
Paper No(s)/Mail Date _____
- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date _____
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: _____

DETAILED ACTION

Examiner's Response to Applicant's Remarks

1. Applicant's arguments filed September 12, 2005 with respect to claims 1,4,6-11,13-17,20-21,23-24 and 26-27 have been considered but are moot in view of the new ground(s) of rejection.

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

3. Claims 1,4,6-11,13-17,20-21,23-24 and 26-27 are rejected under 35 U.S.C. 103(a) as being unpatentable over BEA (WebLogic Server documentation, copyright 1997-2000 and last updated 2/9/2000 and BEA hereinafter) in view of Jacobson et al. (Middleware for Software Leasing over the Internet, ACM, 1999 and Jacobson hereinafter).

As to claims 1,11,17 and 26, BEA WebLogic teaches a method comprising, receiving data requests generated by an application executing on a first system (i.e., "In the XML over HTTP example, StockClient and StockServlet explicitly write XML documents to

the appropriate output stream. In the code segment below StockClient generates XML based on user input that defines a stock trade transaction." ... // send xml to servlet

```
pw.println("<?xml version='1.0' ?>");  
  
// set document type declaration  
  
pw.println("<!DOCTYPE stocktrade SYSTEM '"+serverURL  
  
+ "stocktrade.dtd">");  
  
// set stock trade instructions  
  
pw.print("<stocktrade ");  
  
pw.print("action = "+ (buy ? "'buy' : : "'sell' ");  
  
pw.print("symbol = "+ (webl ? "'WEBL' " : "'INTL' ");  
  
pw.print("numshares = '"+ line + "'");  
  
pw.println(">");"
```

...

"The DOM API provides methods for building and modifying entire XML documents in memory. Once a document has been generated, you simply write the document to the appropriate output stream." The preceding text excerpts clearly indicate that a user most likely through a web based first system/StockClient application sends requests for some stock transaction data to a second system/StockServlet application. StockClient collects all of the necessary request information e.g., buy, sell, stock symbol name, number of shares etc. and generates a structured XML request to send it to StockServlet application through an output stream. The XML document includes elements i.e., data enclosed/delimited by '<' and '>', containing the transaction data.) (Page 4; page 21), generating an Extensible Markup Language (XML)

Art Unit: 2165

structure (page 4) for each data request (page 4) and the XML structure (page 4) including a variable stream (page 4) of data stored in memory (i.e., "The DOM API provides methods for building and modifying entire XML documents in memory) (page 4) of the first system (page 4), the stream (page 4) including an XML element (page 4) for each request (page 4); transferring (i.e., "*StockClient connects to the StockServlet and sends an XML document via the POST method. The following code segment from StockClient obtains the URL to the servlet from the argument list, opens a URL connection, obtains the output stream from the connection, and prints the first couple of lines of the XML document to the output stream.*" The preceding text excerpts clearly indicate that StockClient transfers the request document to the StockServlet.) (page 6) the XML structure (page 4) to a second system (page 4); receiving a response from the second system and updating controls in a graphical user interface based on the response (i.e., "*The most efficient way to handle this is to have your application produce an XML document when responding to a client. Prior to sending the response back to the client, the XML document can then be transformed into HTML or WML depending on the client's browser type. ... In this example, StockServlet acts as a mediator between StockClient and TraderBean. StockClient accepts user input to define a stock trade transaction. This data is used to generate XML in the format defined by stocktrade.dtd and the XML is sent to StockServlet via a POST. StockServlet creates an instance to RequestHandler to parse the XML using a validating SAX parser and operates on TraderBean to execute the trade. StockServlet then receives the result of the trade from TraderBean, generates XML in the format of traderresult.dtd, and sends the XML back to StockClient via a HttpServletResponse. StockClient parses the XML using a validating SAX parser and displays the results to the console.*" The preceding text excerpts

Art Unit: 2165

clearly indicate that the second system i.e., the system/ WebLogic Server with StockServlet, TraderBean etc. receives the transaction request data from the Web based client application StockClient. The second system **parses** the data and then the EJB TraderBean **executes** the data i.e., processes the transaction. The result is then sent back to the first system/ StockClient. StockClient parses the result from the transaction result **and shows to the user in the browser. The browser/graphical user interface is updated to show the response from the second system.**) (page 13; page 21; page 22; page 26; page 30).

BEA does not explicitly teach using a data set object for request and response data.

Jacobsen teaches using a data set object for request and response data (i.e., *"Users interact with the application services through a standardInternet browser, not requiring any additional software. ... The MMM implementation is based on standard Web technologies, such as HTML, XML, and MetaHTML; distributed object computing frameworks, such as CORBA; and database technology, such as ODBC. ... Data Set Objects (DSOs): DSOs represent abstractions for user input and output, as well as input drawn from bulk data providers. DSOs consist of metadata about the data used in computations, such as data format, type, location, size, source, and provider. ... All attributes may be set and read by authorized clients of this object. ... MPOs tie MSOs together with DSOs for input and DSOs for output ..."*

The preceding text excerpts clearly indicate that data set objects (DSOs) are used for input and output. A client can set attributes of the data set object and the data set object (DSO) is used for an input/request to a **distributed application** e.g., CORBA or **EJB application**. Also the DSOs are used for output or response.) (page 1- page 5).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of BEA with the teachings of Jacobsen to include using a data set object for request and response data with the motivation to have support operations for demand-driven data retrieval, data storage, and security management (Jacobsen, page 2, column 2).

As to claim 4, BEA teaches the XML element (i.e. the data is delimited by a tag and called an element) (page 2) is a class object (i.e. instantiated objects of Java classes in Java client or Java server applications e.g. StockClient) (page 5; page 21) whose data is stored to generate XML (page 5; page 21).

As to claim 6, BEA teaches that the data set object (i.e. instantiated objects of Java classes in Java client or Java server applications e.g. StockServlet) (page 5; page 21) includes table dictionaries, column names and data from record sets, and stored procedure parameters (i.e., *"In the XML over HTTP example, StockClient and StockServlet explicitly write XML documents to the appropriate output stream."* ... *"In this example, StockServlet acts as a mediator between StockClient and TraderBean. StockClient accepts user input to define a stock trade transaction. This data is used to generate XML in the format defined by stocktrade.dtd and the XML is sent to StockServlet via a POST. StockServlet creates an instance to RequestHandler to parse the XML using a validating SAX parser and operates on TraderBean to execute the trade. StockServlet then receives the result of the trade from TraderBean, generates XML in the format of traderresult.dtd, and sends the XML back to StockClient via a*

Art Unit: 2165

HTTPServletResponse. StockClient parses the XML using a validating SAX parser and displays the results to the console.” ... “*Session Bean* *Pure business “**logic**” bean *Contain no data *Similar to a **stored procedure**, but able to take advantage of the object services. *Entity Bean* *Represent a row in the table: “data” bean *A **business object** with a unique business identity *Identify determined by the primary key. Combine entity and session Beans as appropriate. An example using an order entry system: *Customer entity bean: the customer record *Order entity bean: the service order *Order process session bean: knows the process for completing an order: 1. Check customer credit limit 2. Check availability of stock 3. Place order, record (commit) a transaction.” ... “*Though ejbStore() is called before the EJB bean is actually written to the database, in the event an **exception** is raised while writing to the database, the framework will assume that the EJB bean has been modified, regardless of the setting of the flag.*” The preceding text excerpts clearly indicate that the second system i.e., the system/ WebLogic Server with StockServlet, TraderBean etc. receives the transaction request data from the StockClient. The second system **parses** the data and then the EJB TraderBean **executes** the data i.e., processes the transaction. The result is then sent back to the first system, i.e., the system with StockClient. StockClient parses the result from the transaction result and shows to the user. Therefore whether it is a stock transaction or it is just getting a stock quote from the server, the TraderBean (whether it is a session or entity bean) has to interact with a persistent storage. Whether the TraderBean is a session or entity bean it performs some business logic for the transaction. The transaction result may be just getting the stock quote from the persistent storage i.e., a database table. The transaction result may comprise **failure/error/exception** to commit the transaction e.g., the requested stock is not available. The result may comprise an exception raised during the

usage of the business logic of the EJBBean. The EJBBean i.e., TraderBean sends the **application result i.e., first data structure** to the StockServlet. The StockServlet converts the result into a **delimited XML structure i.e., second data structure** to the StockClient, which stores application results for the StockClient requested data elements.) (page 4; page 5; page 21; page 22; page 25; page 26; page 30).

As to claim 7, BEA teaches that transferring the XML structure comprises a text transmission protocol (i.e. HTTP) (page 4).

As to claim 8, BEA teaches that the text transmission protocol is Hypertext Transfer Protocol (HTTP) (page 4).

As to claim 9, BEA teaches parsing the XML structure into request statements (i.e., StockServlet parses the XML structure sent by the StockClient) (page 4; page 5; page 21; page 22; page 25; page 26; page 30); and executing the request statements (i.e. StockServlet operates/executes on TraderBean to execute the trade) (page 4; page 5; page 21; page 22; page 25; page 26; page 30).

As to claim 10, BEA teaches translating responses from the executed request statements into an XML format (page 4; page 5; page 21; page 22; page 25; page 26; page 30); and sending the XML formatted responses to the first system (i.e. the StockServlet sends

Art Unit: 2165

the requested result data back to StockClient in XML format for StockClient to process the data or whatever it needs to do with that data) (page 5; page 21).

As to claim 13, BEA teaches causing the second system to parse the stream of text by breaking down the stream of text (page 4; page 5; page 21; page 22; page 25; page 26; page 30) to an executable command format (page 4; page 5; page 21; page 22; page 25; page 26; page 30) utilizing data and parameters related to an application (i.e. StockServlet operates/executes on TraderBean to execute the trade) (page 5; page 21).

As to claim 14, BEA teaches that executing further comprises causing the second system to evaluate the executable commands prior to execution in the second system (i.e. StockServlet operates/executes on TraderBean to execute the trade; The StockServlet or the TraderBean has to look at/evaluate the sent parameters to know what it is supposed to do.) (page 5; page 21).

As to claim 15, BEA teaches that executing further comprises causing the second system to evaluate a result generated by executing the executable commands (i.e. StockServlet operates/executes on TraderBean to execute the trade; The StockServlet or the TraderBean has to look at the sent parameters to know what it is supposed to do with those parameters. After it executes whatever it needs to do, it has to look at/ evaluate the data that it is about to send back to the client) (page 5; page 21).

As to claim 16, BEA teaches causing the second system to convert a result into a stream of text based data in a standardized XML format (i.e. the input stream or output stream through which an application receives the XML data; an application transferring data writes to an output stream and an application receiving data reads data from an input stream.) (page 4); and transmitting the result over the network to the first system (page 5; page 21).

As to claim 20, BEA teaches that the element (i.e. the data is delimited by a tag and called an element) (page 2) is a class object (i.e. instantiated objects of Java classes in Java client or Java server applications e.g. StockClient) (page 5; page 15; page 21).

As to claims 21 and 27, BEA WebLogic teaches a method comprising: in a server, receiving a stream of text-based data in an Extensible Markup Language (XML) format (i.e., *"In the XML over HTTP example, StockClient and StockServlet explicitly write XML documents to the appropriate **output stream**. In the code segment below StockClient generates XML based on user input that defines a stock trade transaction." ... // send xml to servlet*

```
pw.println("<?xml version='1.0' ?>");
```

```
// set document type declaration
```

```
pw.println("<!DOCTYPE stocktrade SYSTEM '"+serverURL  
+ "stocktrade.dtd">");
```

```
// set stock trade instructions

pw.print("<stocktrade ");

pw.print("action = "+ (buy ? "'buy' " : "'sell' "));

pw.print("symbol = "+ (webl ? "'WEBL' " : "'INTL' "));

pw.print("numshares = '"+ line + "'");

pw.println(">");
```

...

"The DOM API provides methods for building and modifying entire XML documents in memory. Once a document has been generated, you simply write the document to the appropriate output stream." The preceding text excerpts clearly indicate that a user most likely through a web browser application/ console application sends requests for some stock transaction data to StockClient application. StockClient collects all of the necessary request information e.g., buy, sell, stock symbol name, number of shares etc. and generates a structured XML request to send it to StockServlet application through an output stream. The XML document includes elements i.e., data enclosed/delimited by '<' and '>', containing the transaction data. Applicant calls this process of creating a structured XML request document in the first system to be sent to a second system through a stream, conversion of application requests.) (Page 4; page 21); parsing the stream into request statements (i.e., *"In the XML over HTTP example, StockClient and StockServlet explicitly write XML documents to the appropriate output stream."* ... *"In this example, StockServlet acts as a mediator between StockClient and TraderBean. StockClient accepts user input to define a stock trade transaction. This data is used to generate XML in the format defined by stocktrade.dtd and the XML is sent to StockServlet via a POST. StockServlet*

Art Unit: 2165

creates an instance to RequestHandler to parse the XML using a validating SAX parser and operates on TraderBean to execute the trade. StockServlet then receives the result of the trade from TraderBean, generates XML in the format of traderresult.dtd, and sends the XML back to StockClient via a HttpServletResponse. StockClient parses the XML using a validating SAX parser and displays the results to the console.” ... “*Session Bean* *Pure business “**logic**” bean *Contain no data *Similar to a stored procedure, but able to take advantage of the object services. *Entity Bean* *Represent a row in the table: “data” bean *A **business object** with a unique business identity *Identify determined by the primary key. Combine entity and session Beans as appropriate. An example using an order entry system: *Customer entity bean: the customer record *Order entity bean: the service order *Order process session bean: knows the process for completing an order: 1. Check customer credit limit 2. Check availability of stock 3. Place order, record (commit) a transaction.” ... “*Though ejbStore() is called before the EJB* is actually written to the **database**, in the event an **exception** is raised while writing to the database, the framework will assume that the EJB has been modified, regardless of the setting of the flag.” The preceding text excerpts clearly indicate that the second system i.e., the system/ WebLogic Server with StockServlet, TraderBean etc. receives the transaction request data from the StockClient. The second system **parses** the data and then the EJB TraderBean **executes** the data i.e., processes or applies some business logic to the input data of the transaction. The result is then sent back to the first system, i.e., the system with StockClient. StockClient parses the result from the transaction result and shows to the user. Therefore whether it is a stock transaction or it is just getting a stock quote from the server, the TraderBean (whether it is a session or entity bean) has to interact with a persistent storage. Whether the

Art Unit: 2165

TraderBean is a session or entity bean it performs some **business logic** for the transaction. The transaction result may be just getting the stock quote from the persistent storage i.e., a database table. The transaction result may comprise **failure/error/exception** to commit the transaction e.g., the requested stock is not available. The result may comprise an exception raised during the usage of the business logic of the EJB. The EJB i.e., TraderBean sends the **application result i.e., first data structure** to the StockServlet. The StockServlet converts the result into a **delimited XML structure i.e., second data structure** to the StockClient, which stores application results for the StockClient requested data elements.) (page 4; page 5; page 21; page 22; page 25; page 26; page 30); and intercepting the request statements prior to execution (i.e., the meaning of intercept in Webster's dictionary is "to gain possession of". Therefore StockServlet has to gain possession of /intercept the request, before StockServlet can actually perform requested job i.e., buy, sell or any other transactions. If the StockServlet does not intercept the request, how can it work on the request? E.g., if the request is sell, StockServlet or TraderBean has to make sure it can be sold or even the requestor has enough stock to sell.) (page 4; page 5; page 21; page 22; page 25; page 26; page 30) and applying additional logic (page 4; page 5; page 21; page 22; page 25; page 26; page 30) based on a type or content of the request (i.e., if it is a buy, the traderBean performs a buy transaction or if it is a sell the traderBean performs a sell transaction.) (page 4; page 5; page 21; page 22; page 25; page 26; page 30).

Art Unit: 2165

As to claim 23, BEA teaches that executing further comprises applying additional logic to responses generated from executing the request statements (page 4; page 5; page 7; page 8; page 21).


As to claim 24, BEA teaches converting responses generated from each of the executed request statements into an XML format (page 4; page 5; page 7; page 8; page 21).

Points of Contact

4. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Apu M. Mofiz whose telephone number is (571) 272-4080. The examiner can normally be reached on Monday – Thursday 8:00 A.M. to 4:30 P.M.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Jeffrey Gaffin can be reached at (571) 272-4146. The fax numbers for the group is (571) 273-8300.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 305-9600.


Apu M. Mofiz
Primary Patent Examiner
Technology Center 2100

October 19, 2005